**Lab 4: Threads**

**Objectives:**

The objectives of this exercise are to demonstrate the creation and management of threads in a Linux environment using the pthread library. First, we will create a new thread utilizing the pthread\_create() function and execute a user-defined function called show(). We will then employ the pthread\_join() function to ensure that the main thread waits for the completion of the created thread, verifying that it handles thread termination appropriately. Additionally, we will illustrate the creation of a thread and subsequently create a child process from within this thread. This exercise also includes demonstrating the versatility of pthread\_create() by passing a function and its argument as parameters, ensuring proper thread execution with arguments. Through these steps, we aim to provide a comprehensive understanding of thread creation, synchronization, and argument passing in multi-threaded programming using POSIX threads.

1. **Creating new thread using pthread\_create()**

ls

mkdir threads

cd threads

nano thread.c

Write following text in nano window.

//basic creation os thread

#include<stdio.h>

#include<pthread.h>

#include<time.h>

void \* show(void \* u){

printf("New thread\n");

}

int main(){

pthread\_t tid;

time\_t t;

time(&t);

printf("\nThis program has been writeen at (date and time): %s", ctime(&t));

pthread\_create(&tid,NULL,&show,NULL);

printf("Main thread\n");

pthread\_join(tid,NULL);

return 0;

}

Press ctrl+x to save file.

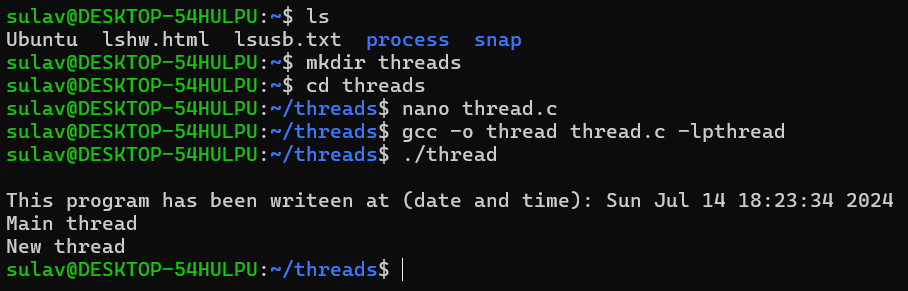
gcc -o thread thread.c -lpthread

./thread

**Interpretation:**

Creating new thread using pthread\_create() function and executing a user-defined function show().

**Output:**

****

1. **pthread\_join() function**:

nano thread\_two.c

//contains two threads

//both calling different function but modifying same global variable

#include<stdio.h>

#include<pthread.h>

#include<stdlib.h>

#include <unistd.h>

#include<time.h>

int value=1;

void \* sleep\_a(void \* u){

printf("New boy\n");

value=value+5;

printf("\nI am going to sleep for %d seconds: ",value);

sleep(value);

printf("\nI slept for %d seconds: \n",value);

}

void \* sleep\_b(void \*u){

printf("old boy\n");

value=value+2;

printf("\nI am going to sleep for %d seconds: ",value);

sleep(value);

printf("\nI slept for %d seconds: \n",value);

}

int main(){

pthread\_t tid,tid2;

time\_t t;

time(&t);

printf("\nThis program has been writeen at (date and time): %s", ctime(&t));

pthread\_create(&tid,NULL,&sleep\_a,NULL);

pthread\_create(&tid2,NULL,&sleep\_b,NULL);

printf("Main thread\n");

pthread\_join(tid,NULL);

pthread\_join(tid2,NULL);

return 0;

}else {

printf("I am the parent.\n");

}

printf("Hello!\n");

return 0;

}

Press ctrl+x to save file.

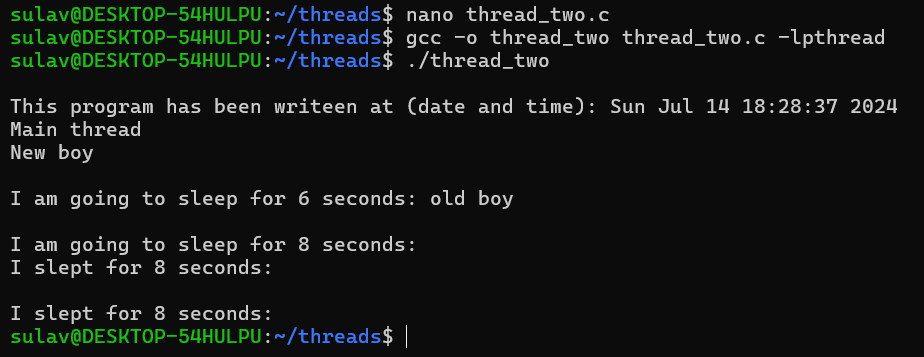
gcc -o thread\_two thread\_two.c -lpthread

./thread\_two

**Interpretation:**

The pthread\_join() function waits for the thread specified by thread to terminate. If that thread has already terminated, then pthread\_join() returns immediately.

**Output:**



1. **Creating thread and creating its child process**

nano thread\_process.c

#include<stdio.h>

#include<pthread.h>

#include<unistd.h>

#include<stdlib.h>

#include<time.h>

void \* show(void \* u){

int pid;

printf("OLD BOY\n");

printf("\nThis is thread its pid is no. %d\n",getpid());

}

int main(){

int pid;

pthread\_t tid;

pthread\_t tid\_child;

time\_t t;

time(&t);

printf("\nThis program has been writeen at (date and time): %s", ctime(&t));

pthread\_create(&tid,NULL,&show,NULL);

printf("Main thread\n");

printf("\nThe pid of main thread is %d: \n",getpid());

printf("\nThe ppid of main thread is %d: \n",getppid());

pid=fork();

if(pid==0){

printf("\nThis is child\n");

printf("\n The id of child is %d\n",getpid());

printf("\n My parent is %d: ",getppid());

exit(0);

}

pthread\_join(tid,NULL);

return 0;

}

Press ctrl+x to save file.

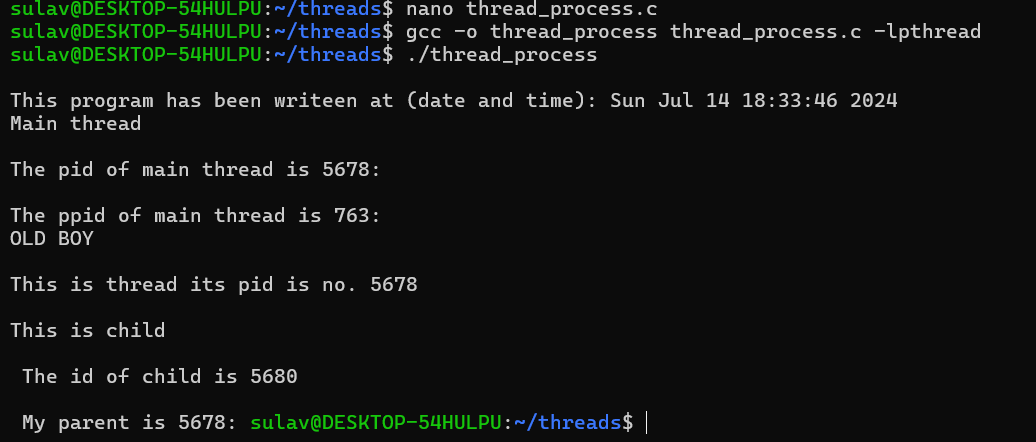
gcc -o thread\_process thread\_process.c -lpthread

./thread\_process

**Interpretation:**

Creating thread and creating its child process

**Output:**

****

1. **Creating thread by passing function and its argument as parameter**

nano thread\_arguments.c

#include <pthread.h>

#include <stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<time.h>

#include<stdlib.h>

int global;

void square(){

global = global\*global;

}

void cube(){

global = global\*global\*global;

}

void \*PrintHello(void \*choice) {

int \*id\_ptr, taskid;

sleep(1);

id\_ptr = (int \*) choice;

taskid = \*id\_ptr;

printf("\nEnter a number: ");

scanf("%d",&global);

if(taskid==0){

square();

printf("\nthe square is %d: \n",global);

}

else if(taskid==1){

cube();

printf("\nthe cube is %d: \n",global);

}

pthread\_exit(NULL);

}

int main( ) {

pthread\_t threads;

int \*choice=malloc(sizeof(int\*));

int rc;

time\_t t;

time(&t);

printf("\nThis program has been writeen at (date and time): %s", ctime(&t));

printf("\n Enter a choice: 0 for square and 1 for cube number: ");

scanf("%d",choice);

printf("Creating thread \n");

rc = pthread\_create(&threads, NULL, PrintHello, (void \*)choice);

if (rc) {

printf("ERROR; return code from pthread\_create() is %d\n", rc);

exit(-1);

}

pthread\_exit(NULL);

}

Press ctrl+x to save file.

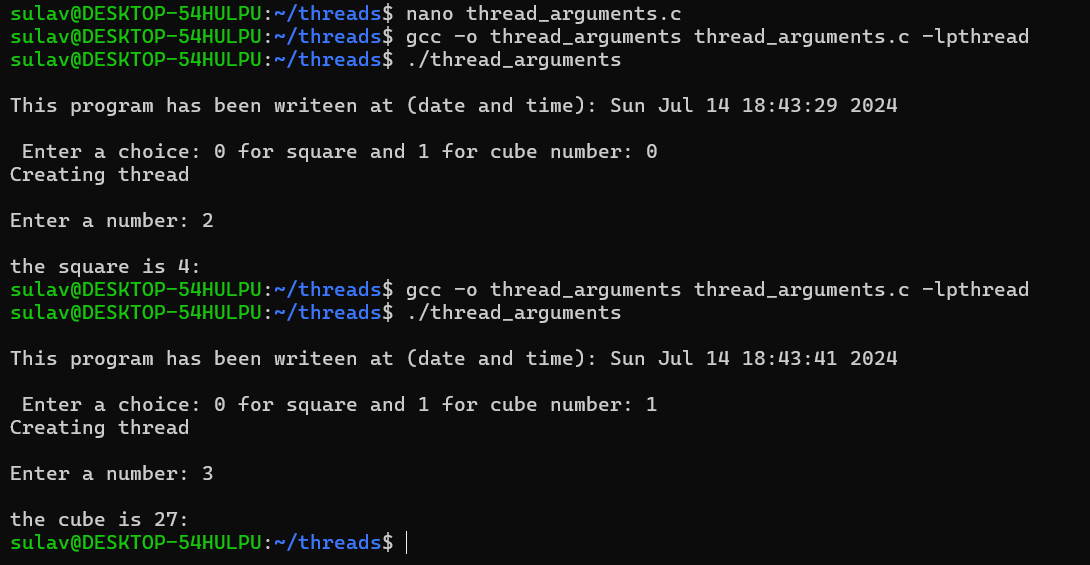
gcc -o thread\_arguments thread\_arguments.c -lpthread

./thread\_arguments

**Interpretation:**

Demonstration of creating thread by passing function and its argument as parameter to the pthread\_create() function.

**Output:**

****

**Conclusion:**

In conclusion, this exercise successfully showcased the fundamental concepts of multi-threaded programming using the POSIX pthread library in a Linux environment. By creating and executing threads with the pthread\_create() function, we demonstrated the ability to run user-defined functions concurrently. The use of pthread\_join() highlighted the importance of synchronization, ensuring that the main thread waits for the completion of spawned threads. Additionally, we explored the capability of creating child processes within threads, extending our understanding of process and thread interactions. The demonstration of passing functions and arguments to threads through pthread\_create() illustrated the flexibility and practical applications of thread management. Overall, these exercises provided a comprehensive understanding of thread creation, synchronization, and communication, equipping us with essential skills for effective multi-threaded programming.